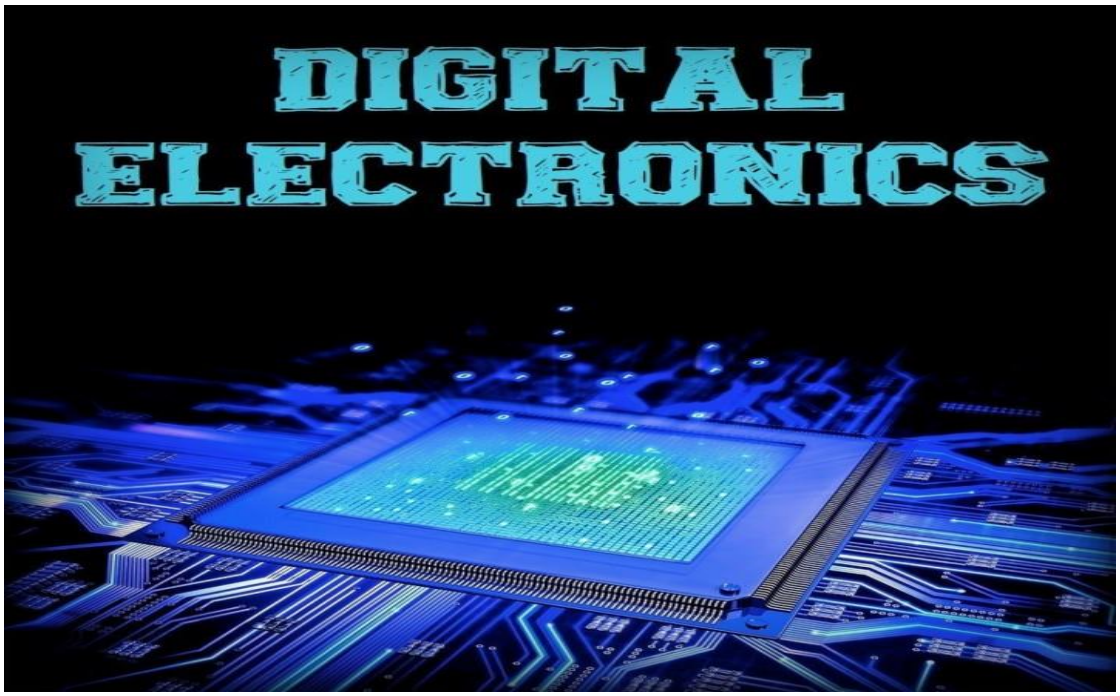
# Abanob Evram

## Assignmen1[Extra]



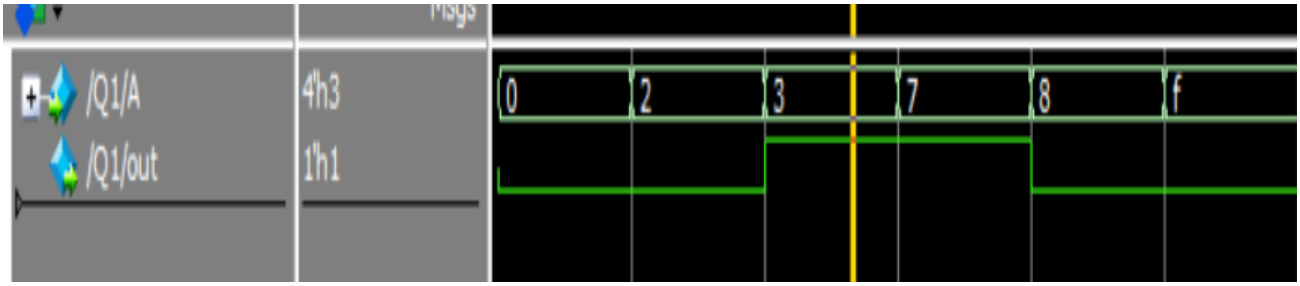
# [Q1]

1) A four-bit binary number is represented as A3A2A1A0, where A3, A2, A1, and A0 represent the individual bits and A0 is equal to the LSB. Design a logic circuit using Verilog that will produce a HIGH output whenever the binary number is greater than 0010 and less than 1000.

- The design takes 1 input **A** (4-bits) and output **out** (1-bit)

## The code:

```
module Q1(A,out);
input [3:0] A;
output out ;
assign out = (2<A&&A<8)?1:0;
endmodule
```

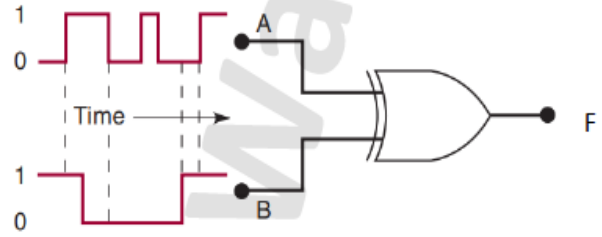


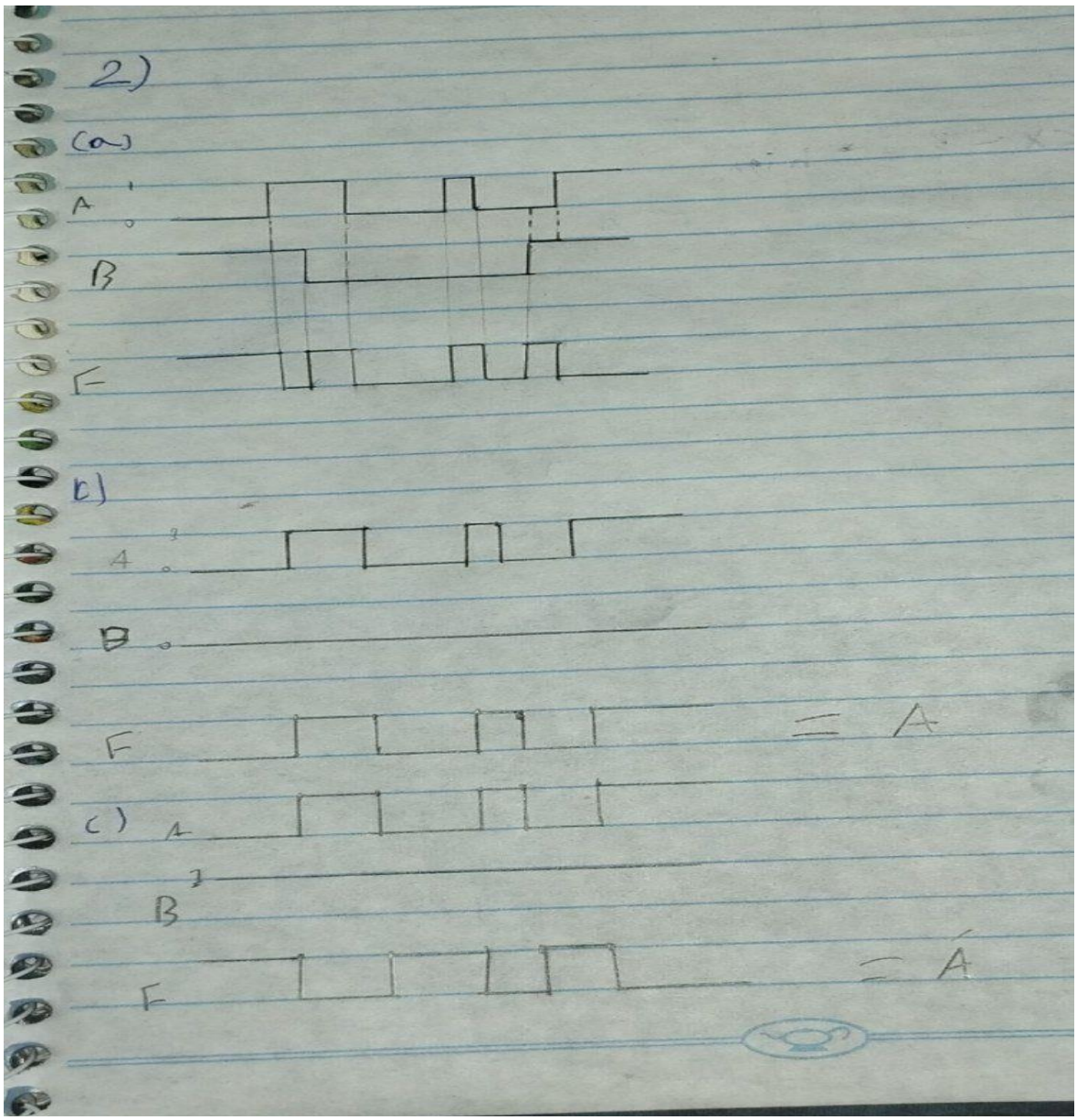
[Q2]

2) (a) Draw with your pen on a piece of paper the output waveform F for the circuit of Figure below.

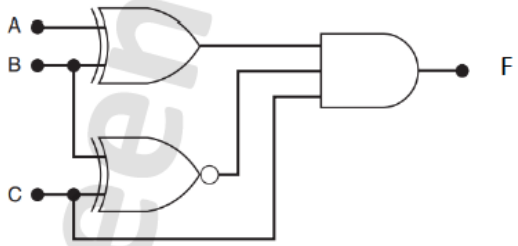
(b) Repeat with the B input held LOW.

(c) Repeat with B held HIGH.



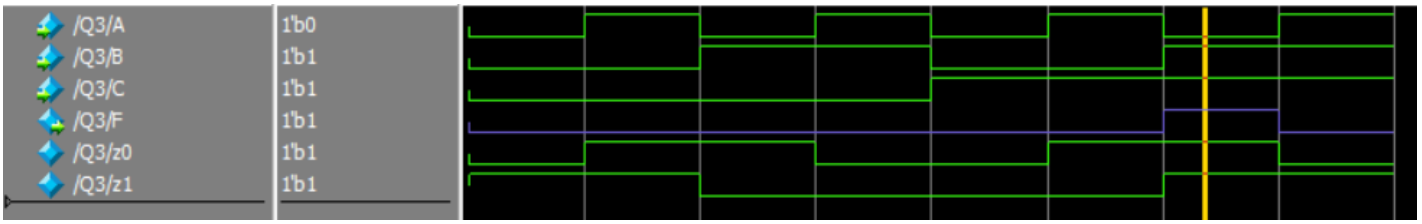
**[Q3]** Design the following circuit using Verilog and determine the input conditions needed to produce F = 1




## The code:

```
module Q3(A,B,C,F);
input A,B,C;
output F;
wire z0,z1;
assign z0 = (A^B) ;
assign z1 = ~(B^C);
assign F=z0&z1&C;
endmodule
```

**Note:** the only case that F=1 it is [A=0,B=1,C=1]



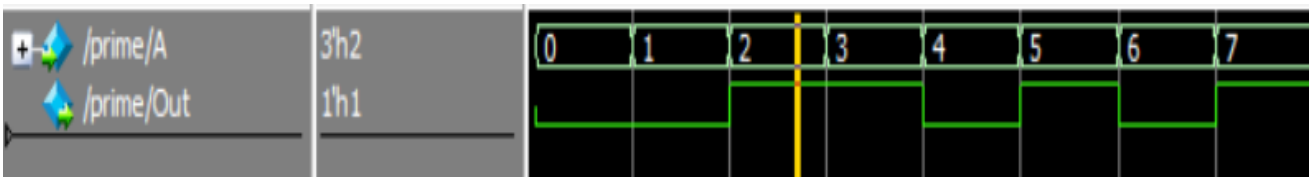
[Q4] 4) Write a Verilog description for a circuit that accepts a three-bit input A and outputs true if the input is a prime number

## The code:

```
module prime(A,Out);
input [2:0] A ;
output reg Out;
always @(*) begin
        case (A)
        2 :Out =1;
        3 :Out =1;
        5 :Out =1;
        7 :Out =1;
        default: Out=0;
endcase
end
endmodule
```

## Another code:

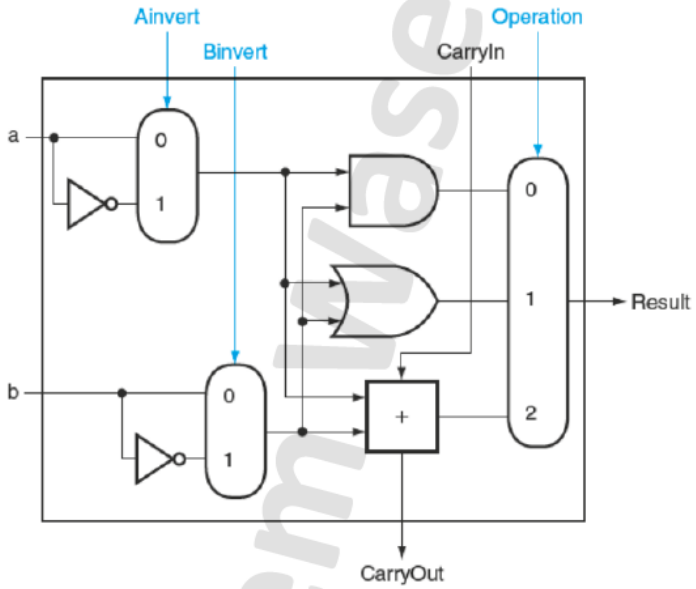
```
module prime (A,Out);
input [2:0] A;
output Out;
assign out=(A==2 ||A==3 ||A==5 || A==7)? 1:0;
endmodule
```



# [Q5]

5) Implement the following 1-bit ALU. If you are unfamiliar with the concept of an ALU, you can find more information by clicking here. Use conditional operator for the multiplexers. For the 3-to-1 Mux, you can use the following format for the conditional operator.

assign <output_signal> = <condition1> ? <value1> : <condition2> ? <value2> : <default_value>);




| Port Name | Type | Size | Description |
|---|---|---|---|
| A | Input | 1 bit | Input a |
| B | | | Input b |
| Ainvert | | | Select signal for the multiplexer to select a or a complement |
| Binvert | | | Select signal for the multiplexer to select b or b complement |
| CarryIn | | | Carry in |
| Operation | | 2 bits | Select signal for the multiplexer to drive the Result output |
| CarryOut | Output | 1 bit | Carry out |
| Result | | 1 bit | Output of the multiplexer |

## The code:

```
module Mux2(In0_Mux2,In1_Mux2,Sel_Mux2,Out_Mux2);
input In0_Mux2,In1_Mux2,Sel_Mux2;
output Out_Mux2;
assign Out_Mux2=(Sel_Mux2==1)?In1_Mux2:In0_Mux2;
endmodule

module Mux3(In0_Mux3,In1_Mux3,In2_Mux3,Sel_Mux3,Out_Mux3);
input In0_Mux3,In1_Mux3,In2_Mux3;
input [1:0] Sel_Mux3;
output Out_Mux3;
assign Out_Mux3=(Sel_Mux3==0)?In0_Mux3:(Sel_Mux3==1)?In1_Mux3:In2_Mux3;
endmodule

module Adder2(In0_Adder,In1_Adder,Carryout_Adder,Carryin_Adder,Out_Adder);
input In0_Adder,In1_Adder,Carryin_Adder;
output Out_Adder,Carryout_Adder;
assign Out_Adder =In0_Adder+In1_Adder+Carryin_Adder ;
assign
Carryout_Adder=(In0_Adder&In1_Adder)|(Carryin_Adder&In1_Adder)|(Carryin_Adder&In0_Adder);
endmodule

module Alu(A,B,Ainvert,Binvert,Carryin,Operation,Carryout,Result);
input A,B,Ainvert,Binvert,Carryin;
input [1:0] Operation;
output Carryout,Result;
wire Z0,Z1,Z2;
Mux2 m1(.Sel_Mux2(Ainvert),.In0_Mux2(A),.In1_Mux2(~A),.Out_Mux2(Z0));
Mux2 m2(.Sel_Mux2(Binvert),.In0_Mux2(B),.In1_Mux2(~B),.Out_Mux2(Z1));
```

Adder2
a1(.In0_Adder(Z0),.In1_Adder(Z1),.Carryin_Adder(Carryin),.Carryout_Adder(Carryout),.Out_Adder(Z2));

Mux3
m3(.Sel_Mux3(Operation),.In0_Mux3(Z0&Z1),.In1_Mux3(Z0|Z1),.In2_Mux3(Z2),.Out_Mux3(Result));

endmodule